Extracting Latent Social Dimensions

Felix Gutmann
felix.gutmann@barcelonagse.eu

Nicholas Halliwell
nicholas.halliwell@barcelonagse.eu

June 27, 2016

Introduction

The focus of this project is around data this is not independent and identically distributed. Most, if not all social media datasets are like this, as there exists dependent relationships between individuals in the network. These networks are said to be multidimensional, given the many ways people can connect. The authors in [1] propose a method to extract unobserved variables in the network, using relational learning to address the dependency issue. These "social dimension" describe how users in the network connect with each other. Essentially the authors are labeling nodes in a network graph, however, nodes can obtain multiple labels. This makes predictions difficult, however, the method of extracting "social dimensions" proposed in [1] were shown to outperform relational learning methods. In this project, we implement this method of "social dimension" extraction, and using the same dataset, attempt to improve their results.

This report has three main sections. First we give a more detailed summary of the key findings in [1]. We then attempt to replicate the authors results and subsequently try to improve their results with a different classifier. Furthermore, we try to extract more informations from the network to boost prediction results.

Summary

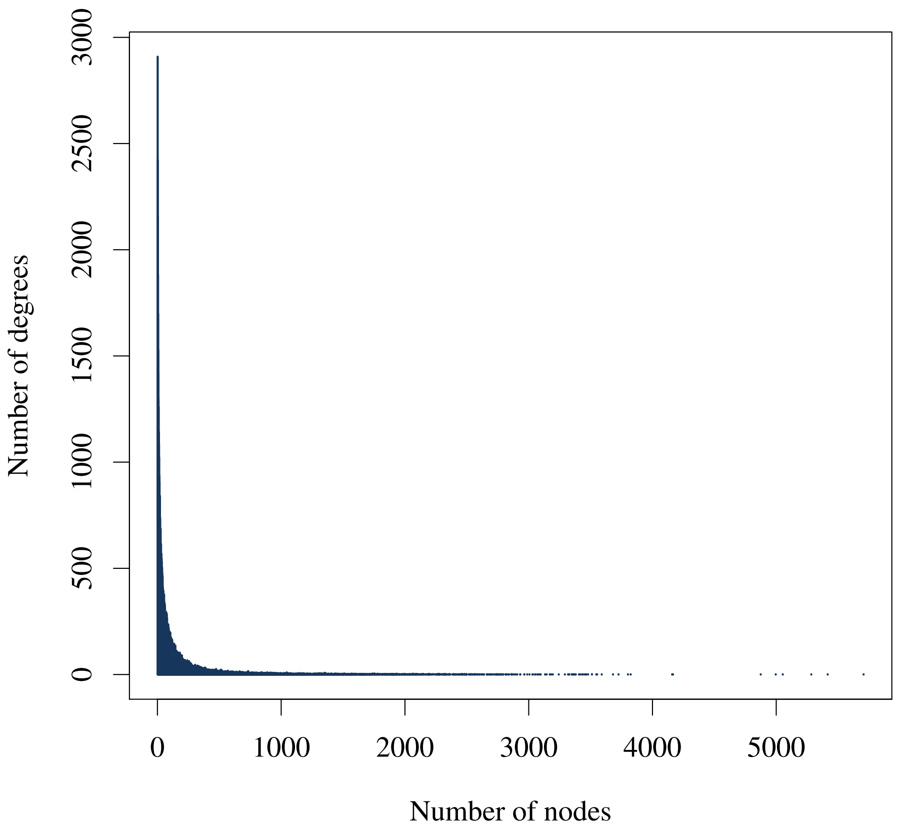
Here we summarize the results of [1] where they propose and implement an algorithm to extract these "social dimensions" and compare classification results on several datasets. First and foremost, the authors are interested in labeling nodes of a social media network where nodes can take multiple labels. The authors give the following example; Actor one connects with Actor two on social media as they work for the same company. Actor two connects with Actor three as they attend the same gym. The data given to us is the interests of Actor one, however, given that Actor two and Actor three connect to Actor one, can we learn the interests of the other actors?

The proposed algorithm starts with a modularity matrix of the graph. After computing this, the algorithm then takes the k largest eigenvectors. The authors recommend k to be between $400 - 600$, we chose 500 for convenience. These eigenvectors are the social dimensions, used as features for supervised multi-label learning. Naturally, computing the modularity matrix for large graphs and the associated eigenvectors is quite costly. The authors provide a Matlab implementation, where the eigenvectors are computed without computing the modularity matrix explicitly.

The authors use a Support Vector Machine using a one-vs-all method and compare classification performance using the Micro-$F1$ and Macro-$F1$ metric. They compare their method to six others, finding their method beats all others in most cases.

The dataset we use comes from Flickr, a website where users upload photos, connect with friends, and tag photos. These tags serve as node labels. This dataset consists of 195 different labels, $80,513$ actors, who are represented as nodes in the graph, and $5,899,882$ links between these actors. We had a more closer look on the network. The graph density is only 0.18 and therefore the network is quite sparse. Figure 1 shows the degree distribution of the Flickr graph

Figure 1: Desgree distribution for full graph



The situation we are dealing with is a multi-label classification problem. The labels are arranged in a binary matrix where an entry for each node is indicating whether a node has a given label. On average each node has 1.4 labels. The label matrix is also very sparse. Only $0.0.69\%$ of the corresponding label matrix are different from zero.

Implementation and Results

We attempt to replicate the results from [1] using the Flickr dataset. We subset this dataset for computational convenience, taking $5 - 15\%$ of the original dataset, and constructing the test set size from $50 - 500$.

We used the authors code for the SVM. After over 8 hours of running, we had no results. We implemented the Support Vector Machine in Python to be more flexible. After sub setting the data, we run the SVM like in [1], however, we were not able to match their results. We constantly get a score of zero. The linear version of the classifier (applied in the paper) depends on on parameter C.[1] The authors do not share their parameter setting they used to tune the algorithm. This could be one possibility for our finding. The tuning of SVM's is very costly. We tried several different parameter settings, without success.

We also implemented a random forest using a one-vs-all method, a classifier known for performing well on large datasets. It serves well for us to produce results in limited time, because it is significantly faster. Even with this classifier, we were not able to match the authors results. We attribute this due to the missing information regarding the size of the test set.[2]

As show we can't replicate the results of the authors. We explored the problems with a lot of different runs. The authors are not very explicit about their setup, they only state the 'training ratio' as input for their model. If we assume they mean they train their classifier on10% of the data set and predict on the remaining 90% their results for such a complex multi-label seem fairly high.[3] Given the fact that our results are so different, we assume that there has to be some different settings, which are not outlined in their work explicitly.

Table 1: Classification results for different test set sizes (training ratio = 10%)

Macro	Micro	Training Size	Test Size
0.01	0.09	8052	50
0.01	0.06	8052	100
0.04	0.12	8052	150
0.02	0.08	8052	200
0.03	0.08	8052	250
0.02	0.07	8052	300
0.02	0.07	8052	350
0.03	0.11	8052	400
0.03	0.09	8052	450

[1]In short the SVM is an optimization problem, where we want to find a hyperplane to separate the data with different labels. If data are not separable the the best hyperplane is tried to be found with minimizing errors. The C parameter controls the penalty for misclassification.

[2]The code for both implementations is available on https://github.com/halliwelln/Networks-Project

[3]The find a score of around 25 %

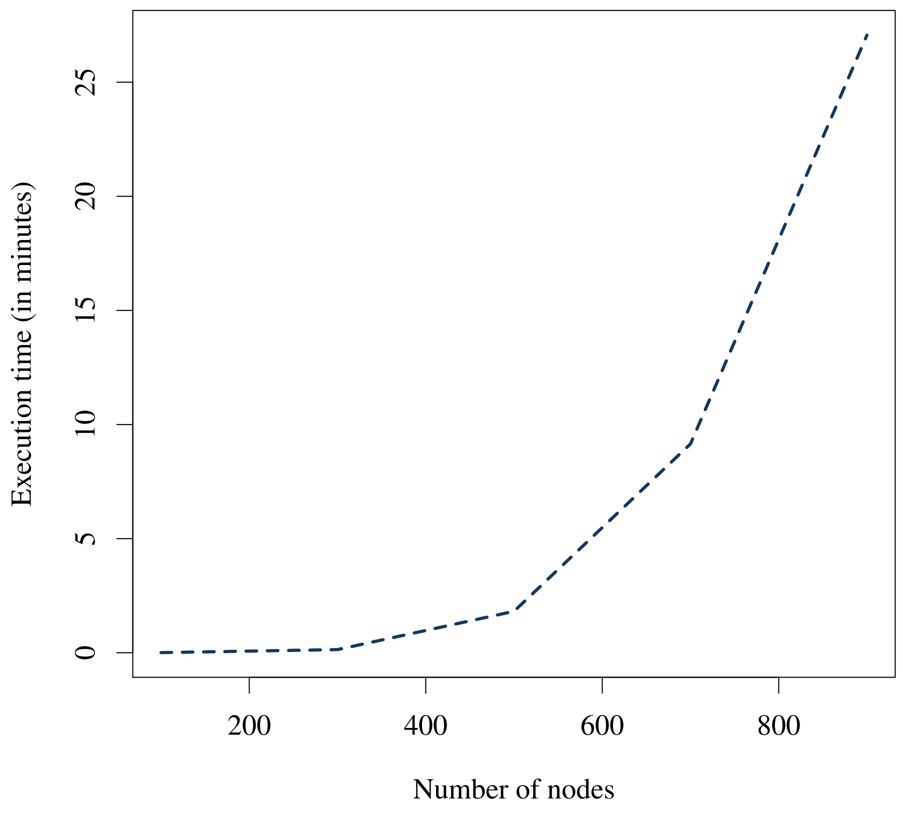
Adding predictors based on network properties

In classification we try to find patterns in the data, in order to discriminate different classes. Therefore, we want to use features (or predictors), which provide class specific patterns in the feature space to make it easier for the classifier to discriminate between those classes. Hence, we want to provide the classifier with information, which helps it to identify such patterns in the data.

The authors state that any additional kind of predictors can be used in the classification setup (e.g. demographics etc.). A natural extension would be to extract more information from the network. However, the extracted information have to help identify what determines nodes affiliation to different tags. The easiest way one could think to add results from a community detection algorithms and also use these as features. We would like to employ algorithms for overlapping community detection, with hopes that this would provide information that nodes with similar tag structure are also of different communities simultaneously. A well known property of social networks is the presence of homophily. Roughly spoken it states that individuals tend to be in similar groups (see. e.g. [2]). Assuming also that similar tend to be associated with the same kind of groups. We want to try if that adds additional pattern structure to the data set.

The downturn is the heavy complexity of our data set. We use the *linkcomm* package of R to find overlapping communities. The main reason is that it works very efficient for a reasonable graph size. Furthermore, the algorithm runs faster if the adjacency matrix is sparse (which in our case is true). A small simulation test indicates that the runtime is increasing heavily with adding new nodes (based on simple random graphs with probability 0.1). Figure 1 shows the execution time in minutes as a function of nodes.

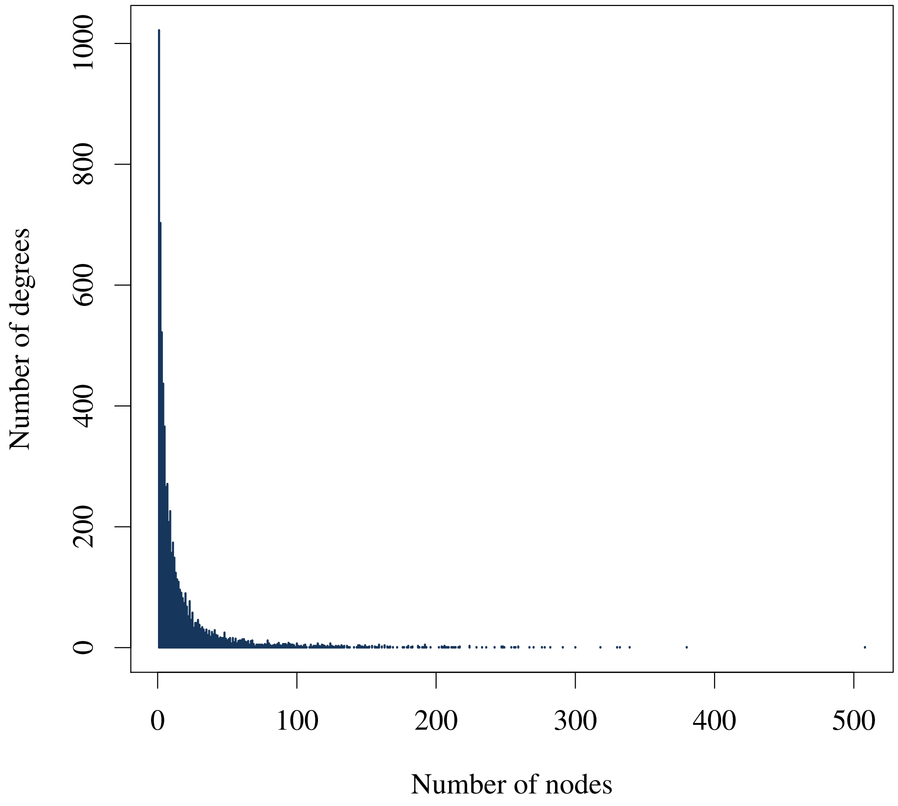
Figure 2: Execution time vs. number of nodes in random graphs



We can take advantage of the sparsity within the Flickr network. Nevertheless we had to reduce the size of the network. In a simple manner we subset uniformly 6819.[4] It turns out that we could keep more or less the basic graph properties like the power law distribution (see figure 3), average number of tags per (1.39) node and graph density (0.27). [5]

[4]The runtime was roughly 45 minutes
[5]We actually sampled 8000, but dropped isolated nodes.

Figure 3: Desgree distribution for random sub graph



We found 611 communities. We found that each node on average is a member of 1.26 communities, which is quite interesting since this seems quite close to average label. Despite the fact we can't match the results for the author's we try to beat our results. We could assume that this would maybe also push up prediction results in the setting of the authors. The following tables show our results. First we extracted the social dimension for our new sub-graph. We then run our classifier on that matrix. Finally we rerun the model with additional dummy variables indicating whether a node belongs to a given community.[6]

Table 2 shows the result for our sub-graph without the additional features. Despite our basic graph properties seemed to be persevered our prediction results dropped down (especially for the macro score). Additionally table 3 shows the result including the new data. We can observe that results are not significantly improving.

[6]This is a large sparse 6819×611 matrix

Table 2: Classification results - sub graph (without communities - training ratio $\approx 75\%$ - 5 repetitions)

Mirco	Macro	Training Size	Test Size
0.006	0.001	5115	1704
0.002	0.000	5115	1704
0.004	0.000	5115	1704
0.002	0.000	5115	1704
0.002	0.000	5115	1704

Table 3: Classification results - sub graph (with communities - training ratio $\approx 75\%$ - 5 repetitions)

Micro	Macro	Training Size	Test Size
0.003	0.000	5115	1704
0.001	0.000	5115	1704
0.006	0.000	5115	1704
0.003	0.000	5115	1704
0.007	0.001	5115	1704

One possible reason is that the new information are not contributing to the model at all. However, there are other possible reasons. First, we add a very big set of predictors to the model. The random forest classifier is doing some kind of feature selection, however given the huge dimension of the model we would have to apply techniques to reduce the predictor space. Furthermore, we would have to tune some model parameters more (number of trees). This on the other hand increases run time significantly, which made it infeasible within our given time frame.

References

[1] Tang, Lei, and Huan Liu. "Relational Learning via Latent Social Dimensions." Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '09 (2009)

[2] Mcpherson, Miller, Lynn Smith-Lovin, and James M. Cook. "Birds of a Feather: Homophily in Social Networks." Annu. Rev. Sociol. Annual Review of Sociology 27.1 (2001): 415-44. Web.